SSSim: a Simple and Scalable Simulator for P2P Streaming Systems

Luca Abeni, Csaba Kiraly, Renato Lo Cigno *
DISI – University of Trento, Italy
{Luca.Abeni;Csaba.Kiraly;Renato.LoCigno}@unitn.it

Abstract

This paper describes SSSim, the **S***imple and* **S***calable* **Sim***ulator for P2P streaming systems. SSSim is designed for performance and scalability, and allows the simulation of the diffusion of a very large number of chunks over large number of peers in reasonable times. This result has been obtained by optimising the simulator for recurrent workloads and minimizing the memory footprint. After describing the simulator's structure and the most important design decisions, the paper presents some experimental results comparing the simulation times of SSSim and a traditional simulator.*

1 Introduction and Related Work

P2P applications and specifically P2P-TV applications are becoming popular and have exited the realm of research oddities and esoteric objects to become an important *technical* and *economic* part of the Internet. Restricting the interest to technical issues, such a transition requires that the design of P2P-TV systems ensures stable and dependable protocols as well as scalable and robust algorithms for the information dissemination.

P2P-TV applications like pplive [2], SopCast [3] or TVAnts [4] are now quite popular on the Internet. Popular however does not mean they are either of high quality or efficient (meaning that they exploit resources well without wasting them).

Although other proposals like SplitStream [9] and ESM [10] exist, most implemented and working applications are based on splitting the video stream into pieces of information, normally called *chunks*, and distributing them to users with (generally proprietary) protocols. Since peers and chunks are selected at run time, the topology of the overlay can be any generic mesh and does not need to have a defined structure with the consequent overhead management. This approach proved extremely resilient to varying network conditions and churn, making it the elective architecture for the design of novel P2P-TV applications.

At the core of distribution protocols lies "smart" algorithms for selecting the peers to exchange information with and the chunk to exchange, algorithms become the main driver of performance and efficiency, and are thus the focus of research and development. A hoard of peer and chunk selection algorithms are being studied and proposed in the literature, most often without proper comparison with other proposals due to the lack of standard and recognized performance evaluation tools.

Even with idealised network models and simplifying assumptions, the system remains so complex to prevent closed form analysis. Apart from some asymptotic results [6], and some performance bounds with assumptions that cannot be met in reality [5], all studies on scheduling algorithms are carried out via simulation ([11], for example). The consequence is often underestimating the impact of *scale*, because simulators are not efficient enough to handle conditions met in real P2P-TV, with particular reference to the stream duration, which in TV should be considered unbounded, since channels are broadcast 24/7.

Several simulation tools are currently under development for the analysis of P2P systems. We briefly overview them here, explaining their different goals with respect to the tool we are proposing.

PeerSim [1] is a java-based simulator for generic P2P and distributed applications. It provides both a cycle-based and an event-driven simulation engine, handling each peer as an instance of a Java Class. Its main purpose and goal is the exploration of messaging protocols, gossiping and epidemic diffusion, and has not been conceived to handle a continuous stream of information, so that it presents some scalability problems especially in streaming dimension (number of chunks), which characterise P2P-TV applications.

The P2PTVSim simulator[1] was originally developed by Politecnico di Torino [11], and we joined this development effort in the context of the Napa-Wine project[2]. This event-driven simulator provides a generic framework for the evaluation of chunk scheduling algorithms over a constrained underlay network. Its design is aimed at evaluating the effect of various network characteristics and impairments on scheduling performance.

Another event-based simulator, OPSS, was presented

*This work is supported by the European Commission through the NAPA-WINE Project (Network-Aware P2P-TV Application over Wise Network – www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412

[1]http://www.napa-wine.eu/cgi-bin/twiki/view/Public/P2PTVSim
[2]http://www.napa-wine.eu

in [7]. This simulator concentrates on the network capacity sharing model, while less emphasis is put on the definition and evaluation of different chunk schedulers.

An extremely scalable tool handling both P2P content distribution and streaming was presented in [8] and used to explore extremely large systems. The tool is based on a semi-analytic approach where a DTMC describing the system is solved via Monte Carlo simulation. This tool however simulates the distribution of stripes in tree or multi-tree structured systems, and the approach cannot be easily used for chunk-based distribution.

The contribution of this work is the description and the availability for the research community of SSSim, an extremely efficient and fast simulator for the exploration of P2P-TV scheduling algorithms with generic topologies. SSSim can scale simulations to large numbers of peers and chunks, allowing the study of algorithms stability in streaming conditions, i.e., when the TV channels is transmitted continuously, like for instance CNN news.

The remaining part of the paper is organised as follows: Section 2 describes the system model simulated by SSSim and introduces the notation used in this paper; Section 3 describes the simulator's requirements explaining why it is worth developing a special-purpose simulator; Section 4 describes the simulator's structure motivating the most important design decisions; Section 5 presents some experimental results comparing SSSim's performance with the performance of P2PTVSim, and Section 6 presents the conclusions.

2 System Model

The development and design of SSSim considers a P2P streaming system that distributes a *timed media* generated by a *source* by dividing the media in chunks having a fixed size. The total number of chunks is indicated as M_c. Each chunk is generated by the source and is distributed to a number of peers which cooperate to the chunks' diffusion.

The system is modelled as a set $\mathcal{S} = \{P_1, \dots P_N\}$ of N peers P_i, plus a special node P_s called *source*. Each peer P_i receives chunks C_j from other peers, and sends them out to other peers at a rate $s(P_i)$. The source, not included in $\mathcal{S}$, generates chunks in order, at a fixed rate λ (C_j is generated by the source at time $r_j = \frac{1}{\lambda} j$).

This paper focuses on a special set of P2P streaming systems: in particular, all the peers are assumed to have infinite download bandwidth and the same upload bandwidth, equal to the source rate. Hence, $\forall i, s(P_i) = \lambda$; the upload bandwidths can be normalised w.r.t. λ, so that $\forall i, s(P_i) = s(source) = \lambda = 1$ and $r_j = j$ (note that this is the limit case to sustain streaming). While this situation might not be very realistic, it is relevant from the theoretical point of view (because the previous definitions allow to easily define some important properties such as the optimality of a chunk diffusion system). Hence, being able to efficiently simulate this kind of systems is important when developing and analysing new P2P scheduling algorithms. In this special case ($\forall i, s(P_i) = \lambda = 1$), at time t the source generates a chunk C_j (with $r_j = t$) and sends it to a peer; moreover, every peer P_i sends a chunk C_h to a peer P_k. All these chunks will be received at time $t + 1$.

When the source generates a chunk C_h, it has to select a *target peer* P_j to which the chunk is sent (this is the *peer scheduling* decision). Similarly, when a peer P_i sends a chunk C_h to a target peer P_j it has to select both the chunk to be sent and the destination peer (these are the *chunk scheduling* and *peer scheduling* decisions). In realistic situations, the target peer P_j is not selected in the set of all the possible peers $\mathcal{S}$, but in a small set of peers known by P_i called *neighbourhood of* P_i. The size of the neighbourhood is indicated as D.

Since the algorithms used for chunk scheduling and peer scheduling heavily affect the chunk diffusion performance, it is very important to evaluate their performance through both simulation and analytical techniques.

3 Requirements

We deem that the most important properties of a chunk/peer scheduling algorithm have to be formally proved; however, simulating the chunk diffusion is often very important to understand the behaviour of an algorithm, to estimate its performance, and to develop new algorithms.

Being the scheduling algorithms the core and engine of P2P systems, a simulator focusing specifically on scheduling, and abstracting to a higher level of approximation the remaining parts of the system is needed. The most important requirements for a such simulator are *simplicity* (so that implementing new features to evaluate particular aspects of a scheduling algorithm is straightforward) and the possibility to *easily implement new scheduling algorithms* in a flexible way. Moreover, *the simulator must be able to simulate the diffusion of a large number of chunks over a large number of peers*. The second requirement is very important, and has two consequences: the amount of memory used by the simulator should not grow proportionally to both the number of peers N and the chunk number M_c, and the simulator's performance (i.e., simulation time) must be measured not only asymptotically (which is normally a property of the simulated algorithm), but on actual simulations.

The constraint on the amount of memory used by the simulator is needed to allow it to properly scale: for example, if a pointer and an integer are needed for every peer-chunk pair (the allocated memory is proportional to NM_c), then simulating the distribution of 20000 chunks over 10000 peers would require at least $1.5GB$ of memory.

The simulator's performance are also important in order to be able to finish a simulation in a reasonable time.

Summing up, the requirements for the simulator are:

- Flexibility,

- Performance (memory and CPU time),
- Simplicity,
- Scalability,
- Possibility to easily add/implement new scheduling algorithms.

As an output, the simulator should be able to produce a trace of the chunk diffusion (when used with a small number of peers/chunks, this feature is useful to check the correctness of new algorithms, and to understand their dynamics and chunk diffusion patterns) and some statistics about diffusion times (used with large numbers of peers/chunks to evaluate the algorithms' performance).

Finally, the simulator should be able to reproduce the chunk diffusion on full meshes (useful for testing the optimality of a scheduling algorithm) or on various kinds of overlays with different properties, both random as n-regular graphs or Watts-Strogats topologies, or mimicking topologies actually built by distributed algorithms. For this latter reason the possibility of reading topologies generated by external tools in standard notation like dot[3]

As most of the existing simulators do not allow to easily implement new scheduling algorithms, and we are not aware of any simulator providing all the requested features in a scalable and efficient way, a new simulator has been developed from scratch: SSSim, the **S**imple and **S**calable **Sim**ulator.

SSSim has been developed for performance and scalability, and most of the design decisions have been taken based on these two main requirements (if scalability and performance are not important requirements, then there are probably better simulators than SSSim around). The simulator has been released under the GPL[4] and is available at `http://imedia.disi.unitn.it/SSSim`.

4 SSSim Structure

SSSim has been originally developed to perform optimality tests on some new scheduling algorithms (named LUc/ELp and Dl/ELp and discussed in [5]), and optimality is defined when all the nodes have the same output bandwidth $s(P_i) = 1$ (see Section 2). Hence, it is not a general-purpose simulator but it is optimised for the previously described system model: at time t, the source generates a new chunk, and every node P_i starts sending a chunk to a target node P_j. All these chunks will be received at time $t + 1$, before starting the new send cycle.

As a result, SSSim has been developed as a *discrete-time simulator* (opposed to an *event-driven simulator*, that would be less efficient in this case, requiring an event queue and all the related overhead). However, the simulator has been designed to be modular and flexible (when this does not affect performance), so most of its modules can be re-used in event-driven simulations.

The simulator is organised in some software modules, interfaced using a clean and well defined API: the overlay generation module, the main loop, the scheduler, and the statistics module.

4.1 Overlay Generation

Each peer P_i is identified by a `struct peer` structure, which contains a pointer `neighbourhood` to an array of pointers to the neighbours. Such an array is filled during the simulation startup time (but can be modified during the simulation to implement churn and dynamic overlays) by the *overlay generation module*. Different kinds of overlays can be used, ranging from a full mesh to a pipeline and passing through various kinds of random graphs.

The full mesh case is special, because it would require to allocate an array of $N - 1$ elements for every peer P_i (and to fill it with with $\mathcal{S} - \{P_i\}$), consuming a large amount of memory (at least $4N(N - 1)$ bytes, depending on the size of a pointer) which is not acceptable for scalable simulations. For this reason, SSSim provides a special graph generator which sets the `neighbourhood` pointer for all the peers to the same array, containing a list of all the peers. In this way each peer P_i will see itself in its neighbourhood (but properly written schedulers can easily cope with this without any performance loss), but the memory requirements goes down to `sizeof(void *)` N. Another special graph generator in the *DOT graph importer*, which generates the overlay by reading the graph description (in the standard DOT language) from a file. This feature allows SSSim to interoperate with other programs: for example, it is possible to import an overlay generated by PeerSim [1] using the newscast algorithm [12].

4.2 Scheduling

A scheduler is described by a `schedule()` function, with prototype

```
void schedule(struct peer *p, int t,
              int *chunk,
              struct peer **target)
```

where `p` is a pointer to the peer which has to send a chunk, `t` is the current time, and the pair (P_i, C_j) selected by the scheduler (target peer and sent chunk) is returned in `chunk` and `target`. This function is completely generic, but some specialised schedulers `schedule_c_p()` (select the chunk first and the target peer after) and `schedule_p_c()` (select the target first) are provided.

The `schedule_c_p()` and `schedule_p_c()` then call a `peer_sched()` function (that can be defined to ELp, RUp, etc..) and a `chunk_sched()` function (that can be defined to LUc, Dl, etc...).

In this way, it is possible to easily modify the chunk or peer scheduler (or, if needed, it is also possible to use a

[3]See the URL: http://www.graphviz.org/ for additional information on dot and graph visualization.

[4]http://www.fsf.org/licensing/licenses/gpl.html

scheduler that selects chunk and peer at the same time). Other helper functions (for utility-based chunk or peer scheduling, random scheduling, etc...) are also provided for convenience; in most of the cases, a new scheduler can be easily obtained by combining existing functions.

4.3 The Main Loop

After initialising the overlay, the simulator sets $t = 0$ and starts running the main loop that:

1. Receives the chunks that have been sent at time $t - 1$
2. Generates a new chunk in the source, invokes the peer scheduler to select a target and sends the chunk
3. For each peer, invokes the `schedule()` function and sends out a chunk
4. Increases t, and returns to step 1

Every time that a chunk is received by a peer, it is added to an array of received chunks in the peer structure. While this approach can simplify the development of the simulator and of new schedulers (and makes statistics gathering very easy), it requires an amount of memory proportional to NM_c, penalising the simulator's scalability. For this reason, SSSim can be configured to use a reduced chunk buffer per peer; in this case, if B is the chunk buffer size then the amount of memory required for a simulation is proportional to NB.

The chunks buffer management can introduce a significant overhead in the simulation. To reduce such an overhead, SSSim inserts the chunks in the buffer according to their generation times, so that chunk C_i generated at time r_i is inserted in position $r_i\%B$ of the buffer. If $B = 2^k$, then $r_i\%B$ can be computed as $r_i\&(B - 1)$, reducing buffer management overhead significantly.

Note that the buffer size B imposes a maximum playout delay (the time after which a chunk is discarded). However, the exact relation between maximum playout delay and B depends on the algorithm used to handle the buffer. The buffer management technique presented above has the advantage that the maximum playout delay is exactly equal to B.

5 Experimental Results

Since SSSim has been designed for performance, the time needed to finish a simulation (in different situations) has been evaluated by repeating a large set of runs on an Intel(R) Pentium(R) D CPU running at $3.40GHz$ with $2GB$ of RAM. All the experiments have been ran simulating a P2P system in which all nodes have the same output bandwidth (equal to the source rate) and have very large download bandwidth. A pretty standard scheduling algorithm (the so called latest useful chunk / random useful peer or LUc/RUp algorithm) has been used for diffusing the chunks.

5.1 Comparison with a Traditional Simulator

In a first set of experiments, SSSim performance have been compared with the performance of P2PTVSim (as an example of a more traditional, event-based simulator), in order to show the improvements due to SSSim's design and optimisations. The results are reported in Table 1.

In a first experiment, the diffusion of 3000 chunks over 1000 peers has been simulated assuming that the peers are connected by a full mesh and have a buffer size $B = 32$. From the table it is possible to notice that SSSim needed less than $1/10$ of P2PTVSim's time to finish the simulation, and consumed about $1/10$ of the memory. In the following experiments, the number of peers N has been increased, showing that SSSim is able to keep under control the amount of memory and execution time needed to finish a simulation (in particular, the maximum amount of memory needed by SSSim seems to scale pretty well). These first results indicate that the optimisations introduced in SSSim to speed up the simulations on fully connected overlays are effective.

Next, the performance of the two simulators have been compared by increasing the playout delay, to test the effectiveness of the optimisations introduced in SSSim chunks buffer handling[5]. To ensure that no chunk is lost, the chunks buffer size has been increased to 3000[6] (the number of chunks), while still maintaining $N = 1000$ (to keep P2PTVSim's simulations times under control). The results in the table show that while the maximum amount of memory needed by the simulators is comparable (because most of the memory is used in the chunks buffer), SSSim is able to finish the simulation in a much shorter time. Since the chunks buffer handling mechanism can introduce some overhead (and waste some memory), when running simulations with "infinite" chunks buffer size it can be useful to disable such a mechanism. SSSim allows to compile the chunks buffer handling code out of the simulator. With this optimisation enabled, the simulation finished in $238s$ and required only $93.461MB$ of memory.

All the simulations mentioned above focused on system configurations for which SSSim has been heavily optimised (full mesh, large buffer sizes, ...). Hence, some additional simulations with reduced neighbourhood size and small playout delays have been run. Again, the results indicate that SSSim's optimizations are effective, allowing it to consume less memory than a traditional simulator and to finish the simulations in a shorter time (in these cases, the differences between SSSim's performance and P2PTVSim's ones are smaller, because SSSim has not been specifically optimised for this setup). Increasing the number of chunks, the time scaled about linearly for both the simulators (for example, 30000 chunks were diffused in $1070s$ by SSSim and in $6625s$ by P2PTVSim), and the

[5]Note that we are using SSSim as a testbed for these optimisations, that could also be ported to other simulators

[6]Note that in SSSim the chunks buffer size must be a power of 2, so a buffer size of 4096 has actually been used

N	M_c	D	B	Time SSSim	Mem SSSim	Time P2PTVSim	Mem P2PTVSim
1000	3000	999	32	$72s$	$2.945MB$	$835s$	$23.473MB$
2000	3000	999	32	$350s$	$3.9MB$	$5609s$	$82.461MB$
3000	3000	999	32	$993s$	$4.957MB$	$17355s$	$179.723MB$
1000	3000	999	3000	$371s$	$101.801MB$	$80282s$	$114MB$
10000	10000	20	32	$355s$	$12.953MB$	$2138s$	$21.148MB$

Table 1. Comparisons between SSSim and a non-optimized simulator (P2PTVSim).

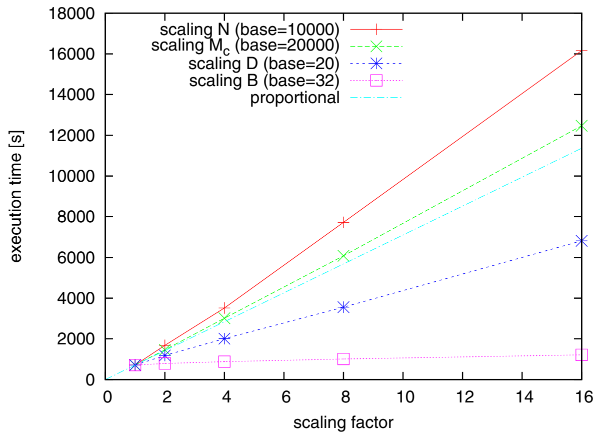

Figure 1. CPU time as a function of N, M_c, D and B.

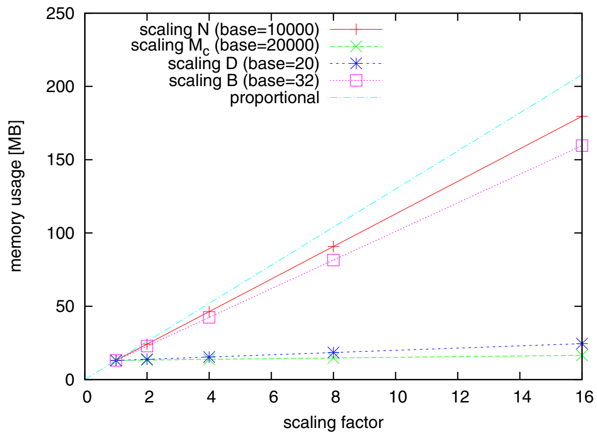

Figure 2. Memory consumption as a function of N, M_c, D and B.

amount of needed memory did not increase too much).

5.2 SSSim Scalability

In a following set of experiments, we examined SSSim's scalability, as a function of four parameters: N, M_c, D and B. Figures 1 and 2 show cpu time and memory consumption as a function of these four parameters. Our reference parameter set is N=10000, M_c=20000, D=20 and B=32, and we scale one parameter at a time. The first curve for example represents simulations varying N from 10000 to 80000 while keeping all other parameters at their reference value.

The figure shows that the execution time is almost proportional to M_c which is the best we could expect since each chunk is diffused individually. With a fixed neighbourhood size, cpu time scales almost proportionally also in N, although a slight extra time is needed due to the longer diffusion time of chunks in the system. Again, we couldn't expect better scaling. Execution time scales better with D, increasing with a small gradient. Increased computation time is due to the use of latest useful chunk selection, which should verify all neighbour's state. Scaling would be even better with different peer scheduling algorithms (such as blind ones). Finally, scaling in B is sub-linear with a very small gradient, which shows the efficiency of the implemented buffer management.

Memory usage scales almost proportionally to both N and B, which shows that the main user of memory is the chunk buffers itself. Other parameters influence memory usage only marginally. More specifically, memory usage does not depend on M_c, which property is very important for being able to simulate streaming systems. Memory usage increase with D is also almost negligible.

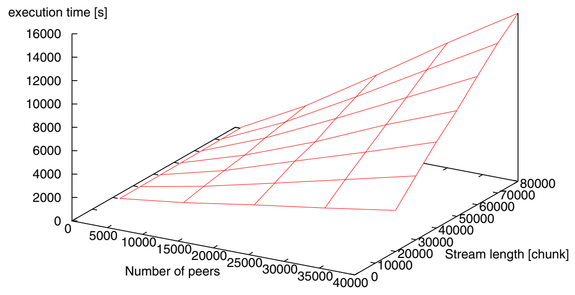

Figure 3. CPU time scaling both N and M_c ($D = 20$ and $B = 32$).

To understand whether these scaling properties hold for a larger set of parameters, we have also run simulations varying both N and M_c. Figures 3 and 4 show both cpu time and memory usage, confirming good scaling characteristics in both parameters.

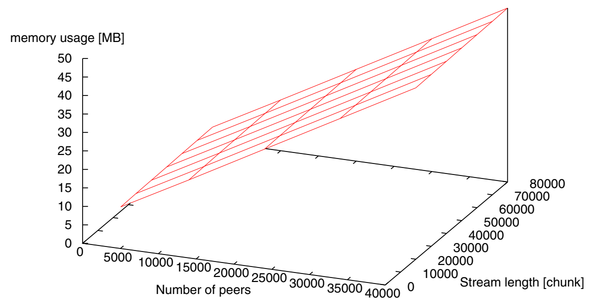

Figure 4. Memory consumption scaling both N and M_c ($D = 20$ and $B = 32$).

6 Conclusions

This paper presented the design and optimisations used in a P2P-TV simulator (named SSSim) to achieve scalability in the execution times and in the amount of used memory. The simulator includes a number of different selection and scheduling algorithms, which lies at the core of the performance evaluation goals of SSSim. Adding new algorithms is easy, since it has been one of the design goals to maintain the definition and inclusion of new algorithms as simple as possible.

The performance of SSSim in terms of memory and CPU usage has been evaluated through an extensive set of experiments comparing it to an event driven simulator (P2PTVSim) developed within the same project framework and optimized for different goals and purposes than SSSim.

SSSim performances scale linearly (in memory and CPU) both in terms of simulated nodes and in terms of simulated chunks, which is the the best possible result given the scenario. Compared to standard event driven simulation tools, the simulation time speed gain can exceed one order of magnitude, enabling the exploration of systems that would otherwise not be feasible. SSSim is released under GPL licence to the entire community.

Acknowledgements

The authors wish to thank Richard Lobb, Ana Paula Couto da Silva, and all the P2PTVSim development team for providing support and allowing scalability and performance comparisons to be carried out.

References

[1] Peersim home page. `http://peersim.sourceforge.net`.
[2] PPLive home page. `http://www.pplive.com`.
[3] SopCast home page. `http://www.sopcast.com`.
[4] TVAnts home page. `http://www.tvants.com`.
[5] L. Abeni, C. Kiraly, and R. L. Cigno. On the optimal scheduling of streaming applications in unstructured meshes. In *IFIP Networking*, 2009.
[6] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg. Epidemic live streaming: optimal performance trade-offs. In Z. Liu, V. Misra, and P. J. Shenoy, editors, *SIGMETRICS*, pages 325–336, Annapolis, Maryland, USA, June 2008. ACM.
[7] L. Bracciale, F. L. Piccolo, D. Luzzi, and S. Salsano. Opss: an overlay peer-to-peer streaming simulator for large-scale networks. *SIGMETRICS Perform. Eval. Rev.*, 35(3):25–27, 2007.
[8] D. Carra, R. L. Cigno, and E. W. Biersack. Graph Based Analysis of Mesh Overlay Streaming Systems. *IEEE Journal on Selected Areas in Communications – JSAC*, 25:1667–1677, December 2007.
[9] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP 03)*, pages 298–313, New York, NY, USA, 2003. ACM.
[10] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast*, 20(8), 2002.
[11] A. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo. A bandwidth-aware scheduling strategy for p2p-tv systems. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing 2008 (P2P'08)*, Aachen, September 2008.
[12] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Technical Report IR-CS-006, November 2003. http://www.cs.vu.nl/globe/techreps.html#IR-CS-006.03.